Computer Science 1 — CSci 1100 — Fall 2020
Exam 3
November 23, 2020

Instructions:

- This take-home exam is worth **100 points**. Given our revised schedule this exam will be due Wednesday, December 9, 2020 at 11:59:59 pm. Please download `exam3_files.zip`. and unzip it into the directory of your exam 3. You will find multiple data files to be used in all parts.
- There are 2 questions on this test. Each question should be answered in a separate .py file and submitted to the correct submission box in the Exam 3 gradeable on Submitty.
- When we test on Submitty,some of the expected outputs and the associated points will be hidden from you, You must test outside of Subnitty to make sure your code is correct. The non-hidden Submitty tests can be used to validate your code, but success there does not guarantee success when we test with the hidden tests.
- Point allocation for each problem is as follows:
  - **Problem 1:** Its worth 60 points out of a total of 100. The distribution of these 60 points is such that there will be 20 points for the visible test cases (on the Submitty autograder), 20 points for the hidden test cases and 20 points for the manual grade by the TA.
  - **Problem 2:** Its worth 40 points out of a total of 100. The distribution for non-hidden, hidden and TA grading points will be 15,15 and 10 respectively.

Honor Statement

There have been a number of incidents of academic dishonesty on homework assignments and exams recently, and this must change. Cases are easily flagged using automated tools, and verified by the instructors. This results in substantial grade penalties, poor learning, frustration, and a waste of precious time for everyone concerned. In order to mitigate this, the following is a restatement of the course integrity policy in the form of a pledge. **By submitting your exm solution files for grading on Submitty, you acknowledge that you understand and have abided by this pledge:**

- **I have not shown my code to anyone in this class, especially not for the purposes of guiding their own work.**
- **I have not copied, with or without modification, the code of another student in this class or who took the class in a previous semester.**
- **I have not used a solution found or purchased on the internet for this assignment.**
- **The work I am submitting is my own and I have written it myself.**
- **I understand that if I am found to have broken this pledge that I will receive a 0 on the assignment and an addiitional 10 point overall grade penalty.**

You will be asked to agree to each of these individual statements before you can submit your solutions to this homework.

Please understand that if you are one of the vast majority of the students who follow the rules and only work with other students to understand problem descriptions, Python constructs, and solution approaches you will not have any trouble whatsoever.

Finally, we are nearly at the end of the semester. So far we have covered a substantial number of ways that your code, however well in runs on your local machine, can fail when we test it on Submitty because of errors or because you failed to follow our directions for proper submissions. **You must test your code and make sure it runs on Submitty to the best of your ability.** We will not re-evaluate code that fails to run, although, non-running code is still eligible for manul grading points.

Problem 1: Amusement Park

In this problem your task is to implement a Tram that operates in an Amusement park (for example Disney in Orlando). You will accomplish this using Python Classes.


Problem Specifications: There are 3 entities in the problem: Park, Tram and Customers. You must create 3 Python Classes that represent these entities. You have a lot of freedom on how you want to handle these entities in terms of the attributes and methods. Using these classes, your code should model the following process:

- The amusement park has a Tram that moves customers from one station to another.
- Every customer gets on the tram starting from a particular station and has a destination station.
- The tram, will always start moving from station number 1(assumed to be the starting point) and will keep moving (making stops at each passing station) until the last station and then return back to its starting point visiting the stations in reverse order.
- Customers get on and off based on their starting location and final destination.
- Each customer can use the tram only once. This means, if the customer moves out of the tram they cannot use it again.
- The simulation stops when all customers have reached their final destination.

Program Execution

- There are json files provided to you in `exam3_files.zip`.Those have the customer and station related information.
- The customer attribute has a list of lists. Each sub-list corresponds to a customer and each element is a starting point and the final destination. For example, in `file1.json`, the first customer boards the tram at station 1 and their final destination is station 5. The second customer boards at station 2 and their destination is station 4 and so on.
- The station attribute has the total number of stations in the park.
- You need to import the `json` module to read these into Python objects. You may use the code shown below:

```
f = open("file1.json")
data = json.loads(f.read())
print(data["Customer"])
print(data["Station"])
```

- Your code will be tested using different files, therefore you must read the file name as user input.
- As the simulation run begins, the tram will start moving from station number 1. the movement from one station to another will be called an iteration and we will count these. At each iteration, there will be customers getting on and off of the tram. Your code must keep track of this because each customer can use the tram once ONLY.

- Finally, we will be printing each iteration of the simulation in a tabular form (see the `exam3_out1` and `exam3_out2` files).
- The header of the table indicates each entity of the simulation. The `*T*` indicates the location of the Tram in each iteration. Customer numbers represent the customers at each station e.g. first customer is represented by number 1, the second by number 2 and so on. Once they get on the tram, they are not visible on the table until they reach their final destination.
- The simulation stops when all customers reach their destinations.

Design of Your Classes

As mentioned earlier, we are allowing you substantial freedom to develop you classes; however, we do have several requirements. Every class must be in its own file. The `Park` class must be in the file `Park.py`, the `Tram` class must be in the file `Tram.py` and the `Customer` class must be in the file `Customer.py`. The main program file for the exam must be in `amusement.py`.

Each class must minimally have an initializer method `__init__()` and a `__str__()` method. We require the following:

- Customer
  - For the `Customer` class, the initializer must take a starting station, a final station, and an identifier.
  - The `str` function should return a string that contains the customer ID and the current position.
- Tram
  - For the `Tram` class, the initializer must take the maximum station number.
  - The `str` function should return a string that contains the current poisition and the number of stations.
  - Additionally, the `Tram` class must have a `move()` method. If `T` is a Tram, `T.move()` should move to the next station on the route.
- Park
  - For the `Park` class, the initializer must take the maximum station number and the customer list of lists from the input `JSON` file. It should create a `Tram` with the appropriate initialization and customers for each sublist in the customer list.
  - The `str` function should return a string that contains the current status of the park and its customers.
- For all classes, the initializers may have additional parameters, but they must have suitable default values.
- Each class must provide its own testing code that is included in the module itself. We will test your modules on Submitty in two ways: by running the modules directly and by importing them into our own file and testing them there.
  - Testing for the `Customer` should create customer 12 with starting location 3 and final destination 7; and then print the customer
  - Testing for the `Tram` should create a tram with 3 stations; print it; call the move method to move it; and print it again.
  - Testing for the `Park` should create a park with 3 tram stations and customers based on `[1,␣3],␣[3,␣1]` and thn print it.

See the file `exam3_module_testing.txt` for an example of this part of the exam.

Problem 2: Movies

In this problem, we are building a recommendation engine for an online subscription platform (like Netflix).

We have test data as a list of 10 customers, such that each of them has given a preference rank of 5 different movies. Assume you have this information saved in a dictionary. Each key refers to a customer/user and its value is a list of ratings for 5 movies. The rating is by position with the customer's ranking for movie 1 being stored at index 0, movie 2 at index 1, etc. User rankings go from 5 (favorite) down to 1 (least favorite). For example the first user (`Customer_3`) ranked movie 3 as their favorite, followed by movies 1, 2, 4 and 5. You can work with the following list for testing your code:

```
Customers={'Customer_3':[4,3,5,2,1],'Customer_5': [1,3,4,2,5],'Customer_1':[5,4,3,2,1],
           'Customer_4':[4,5,2,1,3],'Customer_2':[2,3,4,1,5],'Customer_10':[3,1,2,4,5],
           'Customer_11':[2,5,1,4,3],'Customer_16':[1,2,4,5,3],'Customer_22':[5,1,3,4,2],
           'Customer_100':[3,2,1,4,5]}
```

Part a: For this part you will write a recursive function `ranking_order` that takes in a single customer's preference list and that returns the Dissimilarity Metric defined below. You will also put code in your main to call your function. The main code should take user input corresponding to any of the customers in the dictionary. If the user enters a customer not in the dictionary, ignore it, print a message and exit the program. Use good coding practices. Ignore extra whitespace, but for this program, correct case matters. `Customer_3` should not match `customer_3`.

```
>>>Please enter the customer name =>customer_3
customer_3
Please enter a valid customer name
```

Netflix, has a baseline list (called an optimal list) to compare to each custoner's ranking. The optimal movie ranking (according to their huge database of customers) should be in the order `5,4,3,2,1`. Your function, `ranking_order (L1)` takes a list as input and returns a single value. It **must be a recursive function**, that returns a number which represents the dissimilarity of the user from the optimal list.

Dissimilarity Metric The number returned by the function above is called the dissimilarity metric and is calculated by comparing the user list with the optimal list. It is a simple metric. For every element `i` in the user list, count the elements that are less than or equal to `i` and that occur before the position of `i`. For example, for `Customer_3`, rankings: `4,3,5,2,1`, the ranking of movie 5 (at position 4) is 1 but there is no element in the list that occurs before its position (index 4) and is less than 1 so the dissimilarity metric score for 1 is zero. Now check movie 4, its user ranking is 2 and again there are no smaller elements in the list that occur before its position (index 3). For movie 3, its user ranking is 5 and there are 2 values, `4 and 3` that are less than 5 and occur to the left of it in the list. For movies 2 and 1 with rankings `4 and 3` respectively, again, there are no smaller rankings to the left so both have a score of zero. Overall, this gives us a Dissimilarity Metric of 2 for `Customer_3`. A few sample runs are given here (each one is an independent run):

```
>>>Please enter the customer name =>Customer_1
Customer_1
The dissimilarity metric for this customer is 0
```

```
>>>Please enter the customer name =>Customer_16
Customer_16
The dissimilarity metric for this customer is 8
```

Part b: Write a function `nearest_user(u1,Customers)`, that takes the name of the user `u1`, finds the user(s) nearest to `u1` in the dictionary `Customers` in terms of the dissimilarity score, and returns a set of the closest matches.

Add code to your main to print all closest matches in sorted order along with a recommendation for the movie that the nearest user ranked highest. Your code must consider the case of more than 1 similar users. Nearest users are decided based on the absolute difference between the dissimilarity score of `u1` and that of other users. The user(s) with the smallest difference are considered nearest.

Finally, we will run the entire code (both functions to get a movie recommendation(s)) for a given user as shown below.

In the following sample runs, the final print statement(s) depicts that `Customer_3` has 1 nearest user (`Customer_4`) whose favorite movie is movie number 2.

```
>>>Please enter the customer name =>Customer_3
Customer_3
The dissimilarity metric for this customer is 2
----------------------------------------
Similar User(s) And Recommendations:
----------------------------------------
Customer_4
Customers who chose movie number 3 also chose movie number 2
```

A few more example runs show that if the nearest users have the same highest ranked movie as the current user then the program must print a message with no recommendations:

```
>>>Please enter the customer name =>Customer_4
Customer_4
The dissimilarity metric for this customer is 3
----------------------------------------
Similar User(s) And Recommendations:
----------------------------------------
Customer_22
Customers who chose movie number 2 also chose movie number 1
Customer_3
Customers who chose movie number 2 also chose movie number 3


>>>Please enter the customer name =>customer22
customer22
Please enter valid user name

>>>Please enter the customer name =>Customer_100
Customer_100
The dissimilarity metric for this customer is 7
----------------------------------------
Similar User(s) And Recommendations:
----------------------------------------
Customer_2
No recommendations for this user now

>>>Please enter the customer name =>Customer_5
```

```
Customer_5
The dissimilarity metric for this customer is 8
----------------------------------------
Similar User(s) And Recommendations:
----------------------------------------
Customer_10
No recommendations for this user now
Customer_16
Customers who chose movie number 5 also chose movie number 4
```

All of your code should be in a single file named Movie_recommend.py. As part of testing on Submitty, we will both run your code as submitted, and import your file into our code and test your modules separately. Note that this will allow us to test different `Customers` dictionaries. As an example of this second part see a test run below:

```
$ python
Python 3.6.8 |Anaconda, Inc.| (default, Dec 29 2018, 19:04:46)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE\_401/final)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import Movie_recommend
>>> Customers={'Customer_3':[4,3,5,2,1],'Customer_5': [1,3,4,2,5],'Customer_1':[5,4,3,2,1],
 'Customer_4':[4,5,2,1,3],'Customer_2':[2,3,4,1,5],'Customer_10':[3,1,2,4,5],
 'Customer_11':[2,5,1,4,3],'Customer_16':[1,2,4,5,3],'Customer_22':[5,1,3,4,2],
 'Customer_100':[3,2,1,4,5]}
>>> Movie_recommend.ranking_order(Customers['Customer_16'])
8
>>> Movie_recommend.nearest_user('Customer_4', Customers)
{'Customer_3', 'Customer_22'}
>>>
```